Assignment 1 Reflective Report - Beck Busch

Fetch Plans and Strategies

Attribute	Fetch Type	Label Style	Annotation type	Line	Reasoning
AircraftType seatingZones	eager	explicit	elementCollection	21	SeatingZones are very frequently used and not too memory intensive
Flight bookings	lazy	implicit	OnetoMany	32	Bookings are lazy since they are not needed when querying a flight.
Flight aircraftType	eager	implicit	ManytoOne	35	AircraftType, origin, and destination are all implicitly eager since there is only one object to fetch. These attributes are very commonly used so there is little chance of wasting memory.
Flight origin	eager	implicit	ManytoOne	38	
Flight destination	eager	implicit	ManytoOne	40	
Flight seatPricings	eager	explicit	elementCollection	43	seatPricings were explicitly set to eager since the objects are small and frequently used.
FlightBooking flight	eager	implicit	ManyToOne	18	Flight and user were left as eager since they are important attributes that will be used often.
FlightBooking user	eager	implicit	ManyToOne	20	
FlightBooking seats	lazy	implicit	elementCollection	23	Seats was left as lazy since its an uncommon attribute to use.
User bookings	eager	explicit	OnetoMany	23	Bookings was set to eager since it's something that we use often.

Concurrency Control

To implement concurrency control when booking seats I use pessimistic lock on the flight query. This ensures that any attempted database access during a booking will wait for the booking to finish. If someone attempts to make a booking while another booking is in process then it will wait. If the first booking takes the seats requested by the second booking then the second booking will fail. If the first booking takes longer than 5 seconds then it will fail and the second booking will get the seats.
The granularity is per flight since flight objects are fetched from the database before the seats are checked. If we restructured seats to not be a property of a flight, and instead have a flightId attribute themselves, we could query individual seats and reduce the granularity.

New Features

When it comes to adding new features, there are several things that I would like to add.

Support for modifying an existing booking

Modifying an existing booking could be done quite simply, by fetching the specified booking, making the required changes, then committing to the database. This method would need user auth and the id of the booking that needs to be changed.

Support for dynamic ticket pricing

Dynamic ticket pricing would also be easy to implement. It would consist of an algorithm that takes the base price for a seat and the number of seats remaining, and calculates the new price using a pricing model. This algorithm would be run whenever a booking is made or the available seats are displayed to the user in the frontend. The pricing model would determine the relationship between the number of seats remaining and the price increase.

Support for flight notifications

Flight notifications could be run through a subscription system that starts a AsyncResponce when the user books flight tickets. These responses would be processed every so often (10/30 minutes depending on flight frequency) and if the current time is equal to the flight departure time minus the requested time period, the user gets notified